RDG PLANNING ● DESIGN

RDG Planning & Design, Inc.
301 Grand Avenue
Des Moines, Iowa 50309
www.rdgusa.com

Revit 2022 - XML Parameter Mapping Guide

October 2021

Sean Page, AIA, NCARB, LEED ap

Change Parameter Mapping (21.10.26.01)

Modidfy the Project Parameter column below to map to a new Parameter.

Description	Internal Parameter	Project Parameter
Yes/No Parameter for Ceilings if they are tied to a room instance	RDG_Ceiling_Room_Linked	RDG Linked Ceiling
Text Parameter for Ceilings that is the GUID of the Room instance	RDG_Ceiling_Room_GUID	RDG Linked Ceiling GUID
Yes/No Parameter for Floors if they are tied to a room instance	RDG_Floor_Room_Linked	RDG Lnked Floor
Text Parameter for Floors that is the GUID of the Room instance	RDG_Floor_Room_GUID	RDG Linked Floor GUID
Text Parameter for sorting Sheets in the RDG Browser	RDG_Sheet_Series	RDG Sheet Series
Area Parameter for Rooms based on Occupancy Group	RDG_Square_Foot_Per_Person	RDG Square Foot Per Person
Area Parameter for Rooms to store expected Program Area	RDG_Program_Area	RDG Program Area
Area Parameter for Rooms to store Existing Program Area	RDG_Existing_Area	RDG Existing Area
Double Parameter for calulating Room occupancy counts based on Occupancy Function	RDG_Occupancy_Load_Factor	RDG Occupancy Load Factor
Integer Parameter for overriding the Room occupant load based on Occupancy Function	RDG_Occupant_Override	RDG Occupant Override
Yes/No Parameter for Curtainwall to mark if it has previously been elevated	RDG_Curtain_Wall_Elevation	RDG Curtain Wall Elevation
Integer Parameter for Room to use for Occupancy Load in linked models	RDG_Linked_Occupants	RDG Linked Occupants
Text Parameter for Wall instances to be used with Life Safety Tags and Door Ratings	RDG_Fire_Assembly_Type	RDG Fire Assembly Type
Text Parameter for Door Intances	RDG_Door_Fire_Rating	RDG Door Fire Rating
Text Parameter for Door Instances to identify the WallType they are hosted by	RDG_Host_Wall	RDG Host Wall
Text Parameter for Views and Rooms to identify what Linked Model they were Copied / Associated with	RDG_Link_Name	RDG Link Name
Yes/No Parameter for Room Instances that indicates if they were copied / associated with a	RDG_Linked_Room	RDG Linked Room

Cancel Update

This process enables a developer to use mapped references for custom parameters within Revit. This flexibility allows add-ins to be used with multiple Firms, updated Project Templates, or modified Shared Parameters without the need to refactor or update the code base.

The user in Revit can modify the Project Parameter field shown above to match the current model needs, or update the XML file for a more permanent solution.

More information can be found on my GitHub including a full working example.

RDG Planning & Design, Inc.
301 Grand Avenue
Des Moines, Iowa 50309
www.rdgusa.com

The following XML structure shows a sample of the three parameters that are used for the View Linking tool.

The structure as used in this guide and for the GitHub is this:

1. <ParameterMappings> is the container and is not referenced in the code directly
    1. It is represented by the _parent parameter to the method.
2. <Parameters> is the Parent and is used by the GetParameterMappings() method.
    1. It is represented by the _desc parameter to the method.
3. <Name>, <Parameter>, and <Description> are all Children and are used by the GetParameterMappings() method.
    1. They are represented by the _child1, _child2, and _child3 parameters to the method.
        1. <Name> is fixed value to be referenced in the code
        2. <Parameter> is the name of the Project Parameter to us within the Revit file.
        3. <Description> is to help identify what the parameter is to be used for and where

```xml
<?xml version="1.0"?>
<!--This document is used for mapping parameters within RDG Toolbar for 2022 and above-->
<!DOCTYPE ParameterMappings>
<ParameterMappings>
 <!--Description:  What and where the parameter is used in the code-->
 <!--Name:        The Reference used within the code-->
 <!--Parameter:    The actual name of the Parameter in the Model-->
 <Parameters><!–Parent–>
  <Mapping><!–Descendant–>
   <Description>Text Parameter for Views and Rooms to identify what Linked Model they were Copied / Associated with</Description>
   <Name>RDG_Link_Name</Name>
   <Parameter>RDG Link Name</Parameter>
  </Mapping>
  <Mapping>
   <Description>Yes/No Parameter for View Instances that indicates if they were associated with a Linked Model</Description>
   <Name>RDG_Linked_View</Name>
   <Parameter>RDG Linked View</Parameter>
  </Mapping>
  <Mapping>
   <Description>Text Parameter for View Instances that stored the GUID of the Linked Room</Description>
   <Name>RDG_Linked_View_GUID</Name>
   <Parameter>RDG Linked View GUID</Parameter>
  </Mapping>
 </Parameters>
</ParameterMappings>
```

RDG PLANNING ● DESIGN

RDG Planning & Design, Inc.
301 Grand Avenue
Des Moines, Iowa 50309
www.rdgusa.com

The OnStartup() method of the External Application is used to call the GetDefaultSettings() method.

The GetDefaultSettings() method is then used to call the GetParameterMappings() method and set a static application property to the results of the method. The parameters used in the method are as previously described.

```
using Autodesk.Revit.UI;

namespace RDGRevit.WPF
{
        class App : IExternalApplication
        {
                //This is the Function that tells the Revit application to do something when Revit starts.
                public Result OnStartup(UIControlledApplication RevitApplication)
                {
                        //Call the method to populate the Default Settings for Parameter Mapping or other settings
                        GetDefaultSettings();

                        //Let Revit know it was successfully executed
                        return Result.Succeeded;
                }
                //This is the Function that tells the Revit application to do something when Revit closes.
                public Result OnShutdown(UIControlledApplication RevitApplication)
                {
                        //Let Revit know it was successfully executed
                        return Result.Succeeded;
                }

                private void GetDefaultSettings()
                {
                        ControlParams.ParamMappings = Helpers.Collectors.GetParameterMappings("Parameters",
                                "Mapping", "Name", "Parameter");
                }
        }
}
```

RDG Planning & Design, Inc.
301 Grand Avenue
Des Moines, Iowa 50309
www.rdgusa.com

Create a static application property to store the initial values from the XML file. The ParamMappings property is a collection of custom MapParam class objects that will be defined later.

You can use the automatic / short-hand {get; set;} of the properties and initialize set the default to null on creation.

The additional Version static string Property Version is used to provide the current running version of the application on the top of the Main Window in the User Interface.

```
using System.Collections.Generic;

namespace RDGRevit.WPF
{
	class ControlParams
	{
		//A list of custom MapParam class objects to store the Parameter Mappings
		internal static List<MapParam> ParamMappings { get; set; } = null;
		internal static string Version
		{
			get
			{
				return FileVersionInfo.GetVersionInfo(Assembly.GetExecutingAssembly().Location).FileVersion;
			}
		}
	}
}
```

RDG Planning & Design, Inc.
301 Grand Avenue
Des Moines, Iowa 50309
www.rdgusa.com

The following is a the View Model including the custom Collection Class MapParam with properties for Name, Model and Description that are associated with the child nodes of the XML files.

The INotifyPropertyChanged Interface and NotifyPropertyChanged() method are used in the ParamMapViewModel for the User Interface (UI) component of this tool. Notice that it is only associated with the Model property as that is the only one allowed to be updated by the User directly inside Revit.

The ParamMapViewModel implements the BaseViewModel interface as shown on the next page.

```
using System.ComponentModel;

namespace RDGRevit.WPF
{
        internal class ParamMapViewModel : BaseViewModel
    {
        //List of MapParam classes to hold in the information gathered from the XML and linked to DataGrid
        public List<MapParam> ParamMap { get; set; }
    }

    public class MapParam : INotifyPropertyChanged
    {
        //The Property Changed event is used for ViewModel integration
        public event PropertyChangedEventHandler PropertyChanged;
        //The private property for the Project Parameter
        private string _Parameter;
        //The public property for the Name of the parameter in the code behind
        public string Name { get; set; }
        //The public property for the Project Parameter with the NotifyPropertyChanged event attached
        public string Parameter
        {
            get => _Parameter;
            set
            {
                _Parameter = value;
                NotifyPropertyChanged();
            }
        }
        //The public property for the description of where the parameter is used in the code
        public string Description { get; set; }
        //The method that fires the property changed event to update ViewModel bindings when used
        private void NotifyPropertyChanged(string propertyName = "")
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

RDG Planning & Design, Inc.
301 Grand Avenue
Des Moines, Iowa 50309
www.rdgusa.com

The BaseViewModel class is used to provide PropertyChanged and Command functionality for all other View Models that implement it as a base.

You can also see that PropertyChanged.Fody is also referenced to help create the PropertyChanged events on all public properties. (https://github.com/Fody/PropertyChanged)

```
using PropertyChanged;
using System;
using System.ComponentModel;
using System.Linq.Expressions;
using System.Threading.Tasks;

namespace RDGRevit.WPF
{
	[AddINotifyPropertyChangedInterface]
	public class BaseViewModel : INotifyPropertyChanged
	{
		public event PropertyChangedEventHandler PropertyChanged = (sender, e) => { };

		public void OnPropertyChanged(string name)
		{
			PropertyChanged(this, new PropertyChangedEventArgs(name));
		}

		#region Command Helpers
		protected async Task RunCommand(Expression<Func<bool>> updatingFlag, Func<Task> action)
		{
			if(updatingFlag.GetPropertyValue())
			{
				return;
			}

			updatingFlag.SetPropertyValue(true);

			try
			{
				await action();
			}
			finally
			{
				updatingFlag.SetPropertyValue(false);
			}
		}
		#endregion
	}
}
```

RDG PLANNING ● DESIGN

RDG Planning & Design, Inc.
301 Grand Avenue
Des Moines, Iowa 50309
www.rdgusa.com

The following static method is used for opening and accessing the information within the XML file. In this case, the XML file named ParameterMappings.xml has been located in a sub-directory to the application path named Resources. Using System Reflection we are able to get the base executing location and append with the Path.Combine.

1. Using XDocument via Linq, iterate the XML document looking for the Parent, Descendant, and Child nodes as indicated in the method parameters based on the structure of the XML file.

2. Add a New MapParam collection class to the Params list to be returned

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Xml.Linq;
using Autodesk.Revit.DB;
using Autodesk.Revit.UI;

namespace RDGRevit.Helpers
{
	class Collectors
	{
		internal static List<MapParam> GetParameterMappings(string _parent, string _desc, string
			_child1, string _child2, string _child3 = "Description")
		{
			try
			{
				//List of MapParam elements to store the values retrieved from the XML file
				List<MapParam> Params = new List<MapParam>();
				//Load the XML document from the path relative to the Executing Assembly or this application
				XDocument xDoc = XDocument.Load(Path.Combine(Path.GetDirectoryName(
					Assembly.GetExecutingAssembly().Location),@"Resources\ParameterMappings.xml"));
				//Loop through each descendant of the Parent
				foreach(XElement Elem in xDoc.Descendants(_parent))
				{
					//Loop through each descendant of the first descendant
					foreach(XElement ElemDesc in Elem.Descendants(_desc))
					{
						//Get the child element of each 2nd descendant and the associated value
						Params.Add(new MapParam() { Name = ElemDesc.Element(_child1).Value,
							Parameter = ElemDesc.Element(_child2).Value,
							Description = ElemDesc.Element(_child3).Value });
					}
				}
				//Return the List of MapParam objects
				return Params;
			}
			catch(Exception ex)
			{
				TaskDialog.Show("Parameter Mapping Error", ex.ToString());
				return null;
			}
		}
	}
}
```

RDG Planning & Design, Inc.
301 Grand Avenue
Des Moines, Iowa 50309
www.rdgusa.com

The following is the XAML for the main Windows Presentation Format (WPF) Window User Control.

The DataGrid columns are templated and bound to the ParamMap collection of MapParam items in the View Model. The important part here is that the Model / Parameter cell is editable and with the bindings, any updates made in the Form / Window will be synchronized with the same values in the ParamMap collection of MapParam items through the Property Changed Event handler.

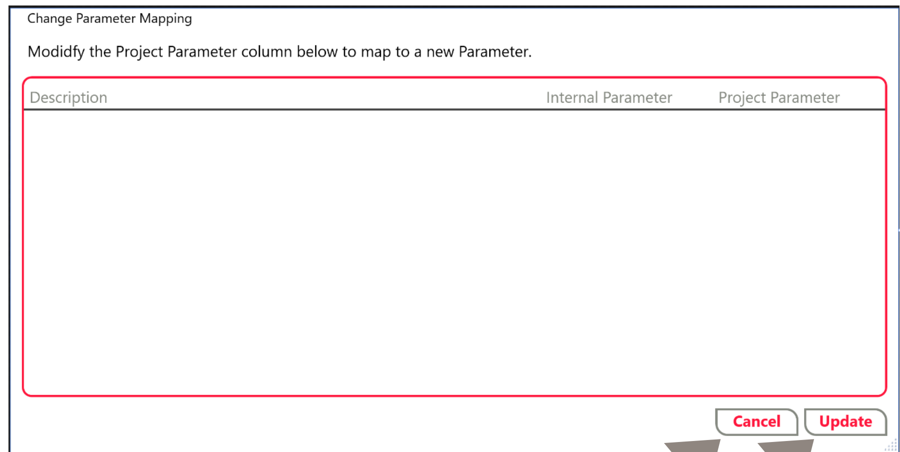
This Two-Way or bi-directional binding allows updates to be synchronized to the data container rather than the items or tags within the interface container or data grid.

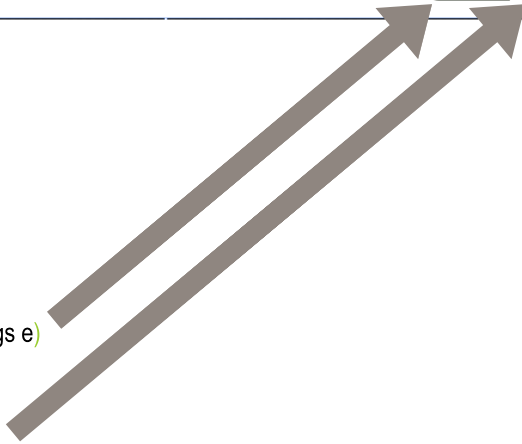
```xml
<Window x:Class="RDGRevit.WPF.ParameterMappingWPF"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:RDGRevit.WPF"
        mc:Ignorable="d" d:DataContext="{d:DesignInstance Type=local:ParamMapViewModel}"
        Title="Parameter Mapping" Width="800" FontFamily="Arial Narrow" ResizeMode="CanResizeWithGrip" Height="400"
    WindowStartupLocation="CenterOwner" Loaded="Window_Loaded" Icon="../Resources/RDG.ico" MinWidth="500" MinHeight="200">
  <Border Padding="10">
    <Grid>
      <Grid.RowDefinitions>
        <RowDefinition Height="26"/>
        <RowDefinition/>
        <RowDefinition Height="35"/>
      </Grid.RowDefinitions>
      <Label Grid.Row="0" Content="Modify the Project Parameter column below to map to a new Parameter:" VerticalContentAlignment="Top"
          Foreground="{StaticResource RDGGreyBrush}" Style="{StaticResource BaseStyle}"/>
      <DataGrid Grid.Row="1" x:Name="dgParamMap" Style="{StaticResource DGrid}" CanUserAddRows="False"
            CanUserResizeRows="False" AutoGenerateColumns="False"
            ItemsSource="{Binding ParamMap}" SelectedValuePath="{Binding Model}">
        <DataGrid.Columns>
          <DataGridTextColumn Header="Description" Binding="{Binding Description}" IsReadOnly="True" Width=".6*"
                      CellStyle="{StaticResource CellReadOnly}" >
            <DataGridTextColumn.ElementStyle>
              <Style>
                <Setter Property="TextBlock.TextWrapping" Value="Wrap"/>
              </Style>
            </DataGridTextColumn.ElementStyle>
          </DataGridTextColumn>
          <DataGridTextColumn Header="Internal Parameter" Binding="{Binding Name}" IsReadOnly="True" Width=".2*"
                      CellStyle="{StaticResource CellReadOnly}" MinWidth="150"/>
          <DataGridTextColumn Header="Project Parameter" Binding="{Binding Model}" IsReadOnly="False" Width=".2*"
                      CellStyle="{StaticResource CellEdit}" MinWidth="150" />
        </DataGrid.Columns>
      </DataGrid>
      <Button x:Name="btnCancel" Content="CANCEL" Grid.Row="3" Grid.ColumnSpan="2" Click="BtnCancel_Click" IsCancel="True"
          Style="{StaticResource RDGButton}" TabIndex="2"/>
      <Button Name="btnUpdateMapping" Content="UPDATE" Grid.Row="3" Grid.ColumnSpan="2" Click="btnUpdateMapping_Click"
          IsDefault="True" Margin="5,5,0,5" Style="{StaticResource RDGButton}" TabIndex="0"/>
    </Grid>
  </Border>
</Window>
```

RDG Planning & Design, Inc.
301 Grand Avenue
Des Moines, Iowa 50309
www.rdgusa.com

The following is the code behind the main Window. The benefits of using a MVVM approach is that there can be very minimal code behind or Events due to the Binding of the properties with the form.

The most important method is the btnUpdateMapping click event. Since we are updating the ParamMap collection through bindings, when the Update button is pressed we just have to set the ControlParams.ParamMapping to the ParamMap collection and then the updated values will be available for use.





```
using System.Windows;

namespace RDGRevit.WPF
{
    public partial class ParameterMappingWPF : Window
    {
        ParamMapViewModel vm = new ParamMapViewModel();
        public ParameterMappingWPF()
        {
            InitializeComponent();
            DataContext = vm;
        }
        private void BtnCancel_Click(object sender, RoutedEventArgs e)
        {
            DialogResult = false;
            Close();
        }
        private void btnUpdateMapping_Click(object sender, RoutedEventArgs e)
        {
            ControlParams.ParamMappings = vm.ParamMap;
            Close();
        }
        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            Title += " (" + ControlParams.Version + ")";
            vm.ParamMap = ControlParams.ParamMappings;
        }
    }
}
```